Correction Using Operators and Decision Constructs - 0

A - Compilation error

This question is on ternary operator (?:). If an expression has multiple ternary operators then number of ? and : should match.

Correction Using Operators and Decision Constructs - 1

B -

```
ANY FRUIT WILL DO
APPLE
MANGO
BANANA
```

“mango” is different from “Mango”, so there is no matching case available. default block is executed, “ANY FRUIT WILL DO” is printed on to the screen.

No break statement inside default, hence control enters in fall-through and executes remaining blocks until the break; is found or switch block ends.

Correction Using Operators and Decision Constructs - 2

D - Compilation error

case values must evaluate to the same type / compatible type as the switch expression can use.

switch expression can accept following:

char or Character

byte or Byte

short or Short

int or Integer

An enum only from Java 6

A String expression only from Java 7

In this case, switch expression [switch (var)] is of byte type.

Correction Using Operators and Decision Constructs - 3

A - Compilation error

Binary plus (+) has got higher precedence than != operator. Let us group the expression.

“Output is:” + 10 != 5

= (“Output is:” + 10) != 5

[!= is binary operator, so we have to evaluate the left side first. + operator behaves as concatenation operator.]

= “Output is: 10” != 5

Correction Using Operators and Decision Constructs - 4

D - -100

First add parenthesis (round brackets) to the given expression: -a++.

There are 2 operators involved. unary minus and Postfix operator. Let's start with expression and value of a.

-a++; [a = 100].

-(a++); [a = 100] Postfix operator has got higher precedence than unary operator.

-(100); [a = 101] Use the value of a (100) in the expression and after that increase the value of a to 101.

Correction Using Operators and Decision Constructs - 5

C - Compilation error

Following are allowed in boolean expression of if statement:

1. Any expression whose result is either true or false. e.g. age > 20
2. A boolean variable. e.g. flag
3. A boolean literal: true or false
4. A boolean assignment. e.g. flag = true

Correction Using Operators and Decision Constructs - 6

C - char var = 7;

D - Character var = 7;

E - Integer var = 7;

switch can accept primitive types: byte, short, int, char; wrapper types: Byte, Short, Integer, Character; String and enums.

In this case, all are valid values but only 3 executes “case 7:”. case is comparing integer value 7.

Correction Using Operators and Decision Constructs - 7

B - Compilation error

As there is no brackets after if, hence only one statement is part of if block and other is outside.

Above code can be written as below:

```
if(grade > 60) {
    System.out.println("Congratulations");
}
System.out.println("You passed");
else
    System.out.println("You failed");
```

There should not be anything between if-else block but in this case, System.out.println(“You passed”); is between if-else and thus Compilation error.

Correction Using Operators and Decision Constructs - 8

B - char var = 10;

C - Short var = 10;

D - Integer var = 10;

F - byte var = 10;

switch can accept primitive types: byte, short, int, char; wrapper types: Byte, Short, Integer, Character; String and enums.

In this case long and double are invalid values to be passed in switch expression. char uses 16 bits (2 Bytes) and its range is 0 to 65535 (no signed bit reserved) so it can easily store value 10.

Correction Using Operators and Decision Constructs - 9

C - 3

a++ == 2 || –a == 2 && –a == 2; [Given expression].

(a++) == 2 || –a == 2 && –a == 2; [Postfix has got higher precedence than other operators].

(a++) == 2 || (–a) == 2 && (–a) == 2; [After postfix, precedence is given to prefix].

((a++) == 2) || ((–a) == 2) && ((–a) == 2); [== has higher precedence over && and ||].

((a++) == 2) || (((–a) == 2) && ((–a) == 2)); [&& has higher precedence over ||].

Let's start solving it:

((a++) == 2) || (((–a) == 2) && ((–a) == 2)); [a=2, res=false].

(2 == 2) || (((–a) == 2) && ((–a) == 2)); [a=3, res=false].

true || (((–a) == 2) && ((–a) == 2)); [a=3, res=false]. || is a short-circuit operator, hence no need to evaluate expression on the right.

res is true and a is 3.

Correction Using Operators and Decision Constructs - 10

C - ANY FRUIT WILL DO

“mango” is different from “Mango”, so there is no matching case available.

default block is executed and as it is the last block inside switch hence after printing “ANY FRUIT WILL DO” control goes out of switch block, main method ends and program terminates successfully.

Correction Using Operators and Decision Constructs - 11

D - AM

There is no compilation error. When Test class is loaded by JVM to invoked main(String []) method, static variable declaration and initialization statement is executed and false is assigned to flag as !true is false.

As java Test AM PM command is used, so args[0] –> “AM” and args[1] –> “PM”.

In ternary operator, boolean expression is evaluated first, !flag evaluates to true and therefore agrs[0] is returned.

AM is printed on to the console.

Correction Using Operators and Decision Constructs - 12

A - Output is: true

“Output is:” + (10 != 5) [Nothing to evaluate at left side, so let’s evaluate the right side of +, 10 != 5 is true.]

= “Output is:” + true [+ operator behaves as concatenation operator]

= “Output is: true”

Correction Using Operators and Decision Constructs - 13

A - Compilation error

case values must evaluate to the same type / compatible type as the switch expression can use.

switch expression can accept following:

char or Character,

byte or Byte,

short or Short,

int or Integer,

An enum only from Java 6,

A String expression only from Java 7.

In this case, switch expression [switch (score)] is of int type.

But case expressions, score < 70 and score >= 70 are of boolean type and hence compilation error.

Correction Using Operators and Decision Constructs - 14

A - a = 18 var = 363

int var = –a * a++ + a– - –a;

int var = –a * (a++) + (a–) - –a;

int var = (–a) * (a++) + (a–) - (–a);

int var = ((–a) * (a++)) + (a–) - (–a);

int var = (((–a) * (a++)) + (a–)) - (–a);

int var = ((19 * (a++)) + (a–)) - (–a); //a = 19

int var = ((19 * 19) + (a–)) - (–a); //a = 20

int var = (361 + 20) - (–a); //a = 19

int var = 381 - (–a); //a = 19

int var = 381 - 18; //a = 18

int var = 363 // a = 18

So,

a = 18

var = 363

Correction Using Operators and Decision Constructs - 15

C - Hello1234

As expression contains + operator only, which is left to right associative. Let us group the expression.

“Hello” + 1 + 2 + 3 + 4

= (“Hello” + 1) + 2 + 3 + 4

= ((“Hello” + 1) + 2) + 3 + 4

= (((“Hello” + 1) + 2) + 3) + 4

[Let us solve it now, + operator with String behaves as concatenation operator.]

= ((“Hello1” + 2) + 3) + 4

= (“Hello12” + 3) + 4

= “Hello123” + 4

= “Hello1234”

Correction Using Operators and Decision Constructs - 16

B - x is equal to 2

Even though default block is available at the top but matching case is present.

So control goes inside matching case and prints “x is equal to 2” on to the console.

After that break; statement takes the control out of the switch- case block.

Correction Using Operators and Decision Constructs - 17

A - Good bonus

$ is valid identifier. $ = 80000

This is an example of ternary operator. First operand ($ >= 50000) is a boolean expression which is true, as 80000 >= 50000 is true.

msg will refer to “Good bonus”.

Correction Using Operators and Decision Constructs - 18

D - 6

Initially i = 5. if(i++ < 6) means if(5 < 6) and then i = 6.

5 < 6 is true, control goes inside if-block and executes System.out.println(i++); This prints current value of i to the console, which is 6 and after that increments the value of i by 1, so i becomes 7.

Correction Using Operators and Decision Constructs - 19

C -

```
MANGO
BANANA
```

fruit refers to String object “Mango”. Matching case is available, MANGO is printed on to the console.

No break statement inside case “Mango”:, hence control enters in fall-through and executes remaining blocks until the break; is found or switch block ends.

So in this case, it prints BANANA and break; statement takes control out of switch block. main method ends and program terminates successfully.

Correction Using Operators and Decision Constructs - 20

C - true

a++ == 3 || –a == 3 && –a == 3; [Given expression].

(a++) == 3 || –a == 3 && –a == 3; [Postfix has got higher precedence than other operators].

(a++) == 3 || (–a) == 3 && (–a) == 3; [After postfix, precedence is given to prefix].

((a++) == 3) || ((–a) == 3) && ((–a) == 3); [== has higher precedence over && and ||].

((a++) == 3) || (((–a) == 3) && ((–a) == 3)); [&& has higher precedence over ||].

Let's start solving it:

((a++) == 3) || (((–a) == 3) && ((–a) == 3)); [a=3, res=false].

(3 == 3) || (((–a) == 3) && ((–a) == 3)); [a=4, res=false].

true || (((–a) == 3) && ((–a) == 3)); [a=4, res=false]. || is a short-circuit operator, hence no need to evaluate expression on the right.

res is true and a is 4.

Correction Using Operators and Decision Constructs - 21

A - int var = ‘7’;

int var = 7; => DEFAULT,

Integer var = 7; => var is of Integer type and case contains char ‘7’. char ‘7’ cannot be compared with Integer and hence compilation error. case ‘7’ can easily be compared with int value but not with Integer type.

int var = ‘7’; => Lucky no. 7

HINT: There is no need to remember. case ‘7’ value means you are trying to equate or compare var (Integer value) with ‘7’ (char).

If assignment operation works then method invocation, switch expression parameter etc. will also work. Integer var = 7; is possible but Integer var = ‘7’; causes compilation error as char cannot be converted to Integer.

Correction Using Operators and Decision Constructs - 22

D - 5

This example is on pass-by-value scheme and very simple to solve. Method m will work on copies and changes done to i and j are local to method m only.

m(++a, a++); [a=3].

m(4, a++); [a=4].

m(4, 4); [a=5].

System.out.println(a); => Prints 5 on to the console.

Correction Using Operators and Decision Constructs - 23

B - 120

Matching case block “case 10:” is found, a = *2; is executed, which means a* = *a* 2; => a = 5 * 2; => a = 10;

No break statement, hence it enters in fall-through.

a = *3; is executed, which means a* = *a* 3; => a = 10 * 3; => a = 30;

a = *4; is executed, which means a* = *a* 4; => a = 30 * 4; => a = 120;

Correction Using Operators and Decision Constructs - 24

C - 11.5 1.0

As floating point numbers are used in the expression, hence result should be in floating point number.

Correct result is:

23 / 2.0 = 11.5

23 % 2.0 = 1.0

Correction Using Operators and Decision Constructs - 25

A - a = 9 res = true

boolean res = a++ == 7 && ++a == 9 || a++ == 9;

= (a++) == 7 && ++a == 9 || (a++) == 9;

= (a++) == 7 && (++a) == 9 || (a++) == 9;

= ((a++) == 7) && ((++a) == 9) || ((a++) == 9);

= ((a++) == 7) && ((++a) == 9) || ((a++) == 9);

= (((a++) == 7) && ((++a) == 9)) || ((a++) == 9);

= ((7 == 7) && ((++a) == 9)) || ((a++) == 9); //a = 8

= (true && ((++a) == 9)) || ((a++) == 9); //a = 8

= (true && (9 == 9)) || ((a++) == 9); //a = 9

= (true && true) || ((a++) == 9); //a = 9

= true || ((a++) == 9); //a = 9

= true; //a = 9

So,

a = 9

res = true

Correction Using Operators and Decision Constructs - 26

A - 10Hello

As expression contains + operator only, which is left to right associative. Let us group the expression.

1 + 2 + 3 + 4 + “Hello”

= (1 + 2) + 3 + 4 + “Hello”

= ((1 + 2) + 3) + 4 + “Hello”

= (((1 + 2) + 3) + 4) + “Hello”

[Let us solve it now,]

= ((3 + 3) + 4) + “Hello”

= (6 + 4) + “Hello”

= 10 + “Hello”

[+ operator with String behaves as concatenation operator.]

= 10Hello

Correction Using Operators and Decision Constructs - 27

C - 11.5 1.0

As floating point numbers are used in the expression, hence result should be in floating point number.

Correct result is:

23 / 2.0 = 11.5

23 % 2.0 = 1.0

Correction Using Operators and Decision Constructs - 28

A - a = 9 res = true

boolean res = a++ == 7 && ++a == 9 || a++ == 9;

= (a++) == 7 && ++a == 9 || (a++) == 9;

= (a++) == 7 && (++a) == 9 || (a++) == 9;

= ((a++) == 7) && ((++a) == 9) || ((a++) == 9);

= ((a++) == 7) && ((++a) == 9) || ((a++) == 9);

= (((a++) == 7) && ((++a) == 9)) || ((a++) == 9);

= ((7 == 7) && ((++a) == 9)) || ((a++) == 9); //a = 8

= (true && ((++a) == 9)) || ((a++) == 9); //a = 8

= (true && (9 == 9)) || ((a++) == 9); //a = 9

= (true && true) || ((a++) == 9); //a = 9

= true || ((a++) == 9); //a = 9

= true; //a = 9

So,

a = 9

res = true

Correction Using Operators and Decision Constructs - 29

A - 10Hello

As expression contains + operator only, which is left to right associative. Let us group the expression.

1 + 2 + 3 + 4 + “Hello”

= (1 + 2) + 3 + 4 + “Hello”

= ((1 + 2) + 3) + 4 + “Hello”

= (((1 + 2) + 3) + 4) + “Hello”

[Let us solve it now,]

= ((3 + 3) + 4) + “Hello”

= (6 + 4) + “Hello”

= 10 + “Hello”

[+ operator with String behaves as concatenation operator.]

= 10Hello

Correction Using Operators and Decision Constructs - 30

C - 26

Initially val = 25.

‘if(val++ < 26)’ means ‘if(25 < 26)’, value of val (25) is used in the boolean expression and then value of val is incremented by 1, so val = 26.

25 < 26 is true, control goes inside if-block and executes System.out.println(val++); This prints current value of val to the console, which is 26 and after that increments the value of val by 1, so val becomes 27.

Correction Using Operators and Decision Constructs - 31

C - false true

Let’s solve the expression at Line n1:

!flag1 == flag2 != flag3 == !flag4

(!flag1) == flag2 != flag3 == (!flag4) //Logical NOT has got highest precedence among given operators

((!flag1) == flag2) != flag3 == (!flag4) //== and != have same precedence and left to right associative, grouping == first

(((!flag1) == flag2) != flag3) == (!flag4) //grouping != next

Above expression is left with single operator ==, whose left side is: (((!flag1) == flag2) != flag3) and right side is: (!flag4). As == is a binary operator, so left side is evaluated first.

((false == flag2) != flag3) == (!flag4) //!flag1 is false

((false == false) != flag3) == (!flag4) //flag2 is false

(true != flag3) == (!flag4) //(false == false) evaluates to true

(true != true) == (!flag4) //flag3 is true

false == (!flag4) //(true != true) evaluates to false

false == true //!flag4 is true

false //(false == true) evaluates to false

Hence, false is printed on to the console.

Let’s solve the expression at Line n2:

flag1 = flag2 != flag3 == !flag4

flag1 = flag2 != flag3 == (!flag4) //Logical NOT has got highest precedence among given operators

flag1 = (flag2 != flag3) == (!flag4) //== and != have same precedence and left to right associative, grouping == first

flag1 = ((flag2 != flag3) == (!flag4)) //grouping == next

Above expression is left with single assignment operator =, whose right side needs to be evaluated first

flag1 = ((false != flag3) == (!flag4)) //flag2 is false

flag1 = ((false != true) == (!flag4)) //flag3 is true

flag1 = (true == (!flag4)) //(false != true) evaluates to true

flag1 = (true == true) //!flag4 is true

flag1 = true //(true == true) evaluates to true

true is assigned to flag1 and true is also printed on to the console

One suggestion: In the real exam, if you find a question containing multiple expressions, then first check if there is any compilation error or not. If there is no compilation error in all the expressions, then only solve the expressions.

Correction Using Operators and Decision Constructs - 32

C - true false

Given statement:

System.out.println((flag = true) | (flag = false) || (flag = true)); //flag = false

System.out.println(((flag = true) | (flag = false)) || (flag = true)); //bitwise inclusive OR | has higher precedence over logical OR ||. flag = false

|| has two operands, Left: ((flag = true) | (flag = false)) and Right: (flag = true). Left operand needs to be evaluated first.

System.out.println((true | (flag = false)) || (flag = true)); //flag = true

System.out.println((true | false) || (flag = true)); //flag = false

System.out.println(true || (flag = true)); //flag = false

|| is a short-circuit operator and as left operand evaluates to true, hence right operand is not evaluated.

Above statement prints true on to the console.

And

System.out.println(flag); prints false on to the console as flag variable is false.

Correction Using Operators and Decision Constructs - 33

C - a = 2, b = 4, c = 7,d = 9, res = false

Given expression:

–a + –b < 1 && c++ + d++ > 1;

–a + –b < 1 && (c++) + (d++) > 1; //postfix has got highest precedence

(–a) + (–b) < 1 && (c++) + (d++) > 1; //prefix comes after postfix

{(–a) + (–b)} < 1 && {(c++) + (d++)} > 1; //Then comes binary +. Though parentheses are used but I used curly brackets, just to explain.

[{(–a) + (–b)} < 1] && [{(c++) + (d++)} > 1]; //Then comes relational operator (<,>). I used square brackets instead of parentheses.

This expression is left with just one operator, && and this operator is a binary operator so works with 2 operands, left operand [{(–a) + (–b)} < 1] and right operand [{(c++) + (d++)} > 1]

Left operand of && must be evaluated first, which means [{(–a) + (–b)} < 1] must be evaluated first.

[{2 + (–b)} < 1] && [{(c++) + (d++)} > 1]; //a=2, b=5, c=7, d=9

[{2 + 4} < 1] && [{(c++) + (d++)} > 1]; //a=2, b=4, c=7, d=9

[6 < 1] && [{(c++) + (d++)} > 1];

false && [{(c++) + (d++)} > 1];

&& is short circuit operator, hence right operand is not evaluated and false is returned.

Output of the given program is: a = 2, b = 4, c = 7, d = 9, res = false

Correction Using Operators and Decision Constructs - 34

F - Compilation error

Line n1 and Line n2 compile successfully.

Let’s check the boolean expression of Line n3:

50 <= score < 60

As multiple operators are available, so let’s group the operators first on the basis of precedence and associativity.

Relational operators (<, >, <= and >=) are at same level and left to right associative, hence given expression can be grouped as:

(50 <= score) < 60

< is a binary operator with two operands: (50 <= score) on the left is of boolean type and 60 on the right is of int type. But < operator is not defined for boolean, int type and hence Line n3 causes compilation error. Line n4 and Line n5 cause compilation error for the same reason.

Correction Using Operators and Decision Constructs - 35

B - true

Given Expression:

i > 3 != false

It has 2 operators > and !=. > has higher precedence over !=, hence given expression can be written as:

(i > 3) != false

Let’s solve above expression:

true != false

true

Hence true is printed on to the console.

Correction Using Operators and Decision Constructs - 36

C - NOT EQUAL 12

Given boolean expression:

(num++ == num++) //num=10

(10 == num++) //Left side operand is evaluated first, value 10 is used in the expression and variable num is incremented by 1, so num=11

(10 == 11) //Right side operand is evaluated next, value 11 is used in the expression and variable num is incremented by 1, so num = 12

Above expression evaluates to false, hence else block is executed and NOT EQUAL 12 is printed on to the console.

Correction Using Operators and Decision Constructs - 37

C - Compilation error

System.out.println(status = false | | status = true | status = false);

As it contains multiple operators, hence let's group the operators first.

System.out.println(status = false | | status = (true | status) = false); //Bitwise inclusive OR | has highest precedence over logical or | | and assignment =

For assignment operator to work, left operand must be variable but in above case, `(true | status) = false` causes compilation failure as left operand (true | status) evaluates to a boolean value and not boolean variable.

Correction Using Operators and Decision Constructs - 38

B - HELLO

Even though compiler is aware that Line n2 will never execute, but it doesn't tag it as unreachable code. Reason for this odd behavior is explained in the Java Language specification:

https://docs.oracle.com/javase/specs/jls/se8/html/jls-14.html###jls-14.21

Following statement results in a compile-time error:

while (false) { x=3; }

because the statement x=3; is not reachable; but the superficially similar case:

if (false) { x=3; }

does not result in a compile-time error. An optimizing compiler may realize that the statement x=3; will never be executed and may choose to omit the code for that statement from the generated class file, but the statement x=3; is not regarded as "unreachable" in the technical sense specified here.

The rationale for this differing treatment is to allow programmers to define "flag" variables such as:

static final boolean DEBUG = false;

and then write code such as:

if (DEBUG) { x=3; }

The idea is that it should be possible to change the value of DEBUG from false to true or from true to false and then compile the code correctly with no other changes to the program text.

Line n2 is not executed but Line n3 executes successfully and prints HELLO on to the console.